Содержание:

Введение

язык программирование бортовой компьютер

Актуальность курсовой работы заключается в том, что в настоящее время, во всем мире, наряду с языками высокого программирования, такими как: Фортран, Алгол, Си, C++, Java и др. особое место занимает Паскаль. Популярности среди программистов он обязан, прежде всего, своей простоте, универсальности и удобству работы в нем. Язык программирования Паскаль используется уже более тридцати лет. На сегодняшний день создано семь версий.

Использование подпрограмм позволяет значительно оптимизировать работу программиста, сократить объем памяти, занимаемый программой, сделать программный код более понятным. Создание пользовательских меню является одной из наиболее важных проблем при разработке пользовательского интерфейса.

Именно поэтому изучение этого аспекта программирования является особенно актуальным при написании современных программ на языках высокого уровня.

Объектом исследования курсовой работы являются языки программирования высокого уровня. Первая глава нашей работы посвящена сравнительному анализу наиболее распространенных языков, их классификации, описанию их достоинств и недостатков.

Предметом исследования данной курсовой работы является изучение такого актуального вопроса, как использование процедур, функции и подпрограмм в языках программирования высокого уровня, а также основных подходов к созданию пользовательских меню.

Таким образом, **целью** данной работы является раскрытие теоретических аспектов рассматриваемой темы «Классификация языков программирования высокого уровня», а также приобретение практических навыков использования процедур и функций на примере реализации конкретной задачи.

При выполнении курсовой работы пред нами были поставлены следующие **задачи:**

- рассмотреть особенности и произвести квалификацию языков программирования высокого уровня.

- дать понятие высокоуровневых языков программирования

- раскрыть процедурно-ориентированные языки;

- рассмотреть проблемно-ориентированные языки

- изучить объектно-ориентированные языки.

В ходе **теоретического исследования** нами были использованы следующие основные литературные источники: Кнут Д. Искусство программирования для ЭВМ. Т1. Основные алгоритмы. М.: Мир, 2014. Прайс Д. Программирование на языке Паскаль: Практическое руководство. Пер. с англ. - М.:Мир, 2013. Фаронов В.В. Турбо-Паскаль 7.0. Начальный курс. М.: "Нолидж", 2013. Культин Н.Б. Программирование в Turbo Pascal 7.0 и Delphi. СПб.:БХВ - Санкт-Петербург, 2012. А также другие источники, перечень которых приведен в списке использованной литературы.

Структура работы: данная курсовая работа состоит из введения, двух глав, заключения и списка использованной литературы.

1. Обзор и анализ особенностей применения языков программирования

1.1 Особенности применения языков программирования высокого уровня

Языки программирования высокого уровня используют в аппаратно-независимых системах программирования.

Языки программирования высокого уровня подразделяют на [3]:

- процедурно-ориентированные;
- проблемно-ориентированные;
- объектно-ориентированные.

Каждый из описанных ниже языков программирования применяется для решения определенного круга задач.

К первому классу языков, который используется для записи процедур или алгоритмов обработки информации относят:

а) язык Фортран (Fortran). Является одним из первых языков программирования высокого уровня. К его основным достоинствам относится наличие огромного числа математических библиотек, поддержка работы с целыми, вещественными и комплексными числами высокой точности [3,4], встроенных средств обработки массивов.

б) язык Бейсик (Basic). Был разработан в 1964 г. в качестве языка для обучения программированию [12].

Основными достоинствами этого языка являются, простой синтаксис, который позволяет в кратчайшие сроки освоить этот язык программирования, простота реализации графического интерфейса, возможность использования WinAPI функций, что значительно расширяет возможности языка.

Одним из основных недостатков языка является то, что он поддерживает только операционные системы семейства Windows, DOS и Mac OS X, что значительно сужает сферы его применения.

в) язык Си (C) был создан в 1969-1973 годах в качестве языка системного программирования и первоначально предназначался для написания ОС UNIX [4,12]. В 1980-е гг. язык С был дополнен инструментами объектно-ориентированного программирования и на основе него был создан язык С++.

Одним из главных достоинств является кроссплатформенность, а также минимальные аппаратные требования для запуска скомпилированных программ, широкий набор средств для реализации как прикладных, так и системных задач.

К недостаткам языка можно отнести отсутствие четкой стандартизации. В ходе исторического развития языка его элементы зачастую заимствовались из других языков, вне зависимости от наличия других элементов. Это привело к наличию дублирующих и иногда противоречащих друг другу элементов. Данные аспекты привели к тому, что язык стал чрезвычайно сложным для восприятия.

г) язык Паскаль (Pascal). Был создан математиком Н. Виртом специально для обучения программированию. Однако со временем стал широко применяться для

разработки программных средств в профессиональном программировании.

Самая первая версия была создана в 1968 году профессором кафедры вычислительной техники Швейцарского федерального института технологии Никласом Виртом [3]. Основной целью, при создании нового языка, является его простота, с сохранением всех достоинств уже имеющихся языков высокого уровня программирования.

Популярность созданного языка стала столь высокой, что уже к 1980 году насчитывалось более восьми десятков его трансляторов. В начале 80-х годов язык программирования Паскаль еще более усилил свои позиции после создания трансляторов Turbo-Pascal для персональных компьютеров. С этого момента язык смело вышел за рамки узкого использования программистами-профессионалами. Он начал использоваться как рабочий инструмент пользователей и как средство обучения языков программирования.

Одним из главных достоинств языка Паскаль является четкая структуризация, удобная среда разработки и отладки, позволяющая пользователю обнаружить логические и синтаксические ошибки в программе. Также к достоинствам можно отнести высокую скорость компиляции программ, возможность использования вставок языка Ассемблер.

В отличие от языка С (С++) в при использовании Паскаль сведены к минимуму возможные синтаксические неоднозначности [1,2], синтаксис языка является интуитивно понятным и доступным, поскольку, как уже было отмечено выше, язык изначально разрабатывался для обучения студентов программированию.

К недостаткам первоначально разработанного компилятора можно бы отнести ряд ограничений: невозможность передачи функциям массивов переменной длины, ограниченная библиотека ввода-вывода, отсутствие средств для подключения функций написанных на других языках и раздельной компиляции [7,9].

К проблемно-ориентированным относят следующие языки программирования:

а) язык Лисп. Считается вторым после Фортрана старейшим высокоуровневым языком программирования [3,16]. Данный язык наиболее часто применяется при разработке экспертных систем и систем аналитических вычислений. Существуют современные версии этого языка, которые активно применяются при разработке новейших web-технологий. Также модификации данного языка используются в качестве встроенных языков программирования в САПР. Примером может

послужить AutoLISP - язык для разработки надстроек в продуктах компании AutoDesk.

б) язык Пролог. Используется для реализации систем искусственного интеллекта, а также и интеллектуальных систем баз данных [3].

Написание программ на языке Пролог существенно отличается от использования других языков программирования. Программа на Прологе не является реализацией некоторого алгоритма, а представляет собой запись на языке формальной логики [16]. Таким образом, данный язык относится к описательным языкам программирования.

Таким образом, сферой применения данного языка является решение логических задач. Для создания вычислительных, графически задач, реализации пользовательского интерфейса данный язык не предназначен.

В настоящий момент наиболее активно используются и развиваются следующие среды программирования [3,4]:

а) Delphi (Lazarus некоммерческая - версия для ОС семейства Linux) - основана на Object Pascal;

б) C++, C# (~ C);

в) Visual Basic (~ Basic);

г) Visual Fortran (~ Fortran);

д) Prolog++ (~ Prolog).

Предметом исследования данной курсовой работы является изучение такого актуального вопроса, как использование процедур, функции и подпрограмм в языках программирования высокого уровня, а также основных подходов к созданию пользовательских меню.

Одним из важных факторов для выполнения поставленной задачи может отказаться и верный выбор языка программирования. Для реализации поставленной задачи можно использовать различные языки высокого уровня, наиболее распространенными из которых являются C++ и Паскаль.

Проанализировав все выше изложенное, нами был сделан аргументированный выбор языка программирования высокого уровня Паскаль для дальнейшего

изучения предмета исследования.

1.2 Современный язык программирования высокого уровня

В условиях динамично меняющегося мира, когда еще вчера уважаемые профессии превращаются в ничто, многие люди ищут чем заняться в жизни, чтобы это было и интересно и актуально нынешнему времени, и в то же время прибыльно. Очень часто подобные поиски приводят к программированию: хорошие программисты даже в СНГ зарабатывают тысячи долларов, имеют немало свободного времени, возможность работать удаленно и имеют шансы на карьерный рост.

Помимо указанных преимуществ программирование отличается и тем, что для его освоения совсем не нужно тратить годы времени, протирая штаны в университетах. Здесь все решает самообразование, в интернете есть все необходимые материалы для успешного самообучения любому языку программирования: уроки в текстовом виде, видео уроки, инструкции, советы от опытных специалистов и прочие образовательные материалы. Таким способом можно легко освоить современные технологии программирования и найти по-настоящему достойную работу.

Но перед тем, как взяться за дело, нужно ответить себе на один важный вопрос: какие современные языки программирования в 2017 году будут актуальны, на какой из их тратить свое время и усилия? От правильно ответа здесь зависит очень много – сложность и скорость процесса обучения, минимальный порог вхождения в реальную деятельность, дальнейшие перспективы в карьерном плане.

Чтобы определиться с конкретным языком программирования для изучения (одним или несколькими) нужно для начала ответить себе на вопрос: в какой области программирования человеку хотелось бы работать. Наиболее популярными и бурно развивающимися сферами сейчас выступают:

Технологии web программирования. Создание сайтов, онлайн сервисов и банкингов, интернет магазинов, лендинг страниц для бизнеса и тому подобных вещей – все это является частью веб-программирования. Все больше людей на планете становятся пользователями интернета, он становится все более быстрым и дешевым, активными пользователями сети становятся даже люди зрелого и преклонного возраста. В 2017 году эта тенденция только усиливается и потолка ее

развития еще не видно. Поэтому очень выгодно изучать языки программирования связанные с интернетом. Список самых популярных из них будет представлен ниже;

Программирование для гаджетов: смартфонов, планшетов, смарт часов, очков виртуальной реальности и прочее. Это также очень интересная и быстро развивающаяся сейчас область деятельности. Миллиарды людей во всем мире пользуются гаджетами, читают на них новости, смотрят видео, слушают музыку и занимаются массой других вещей. Все эти функции электроники возможны благодаря хорошо написанному программному обеспечению. Часто оно стоит намного дороже, нежели само физическое устройство. Программировать в этой сфере сейчас очень выгодно. Стоит отметить, что здесь чаще всего применяются языки программирования высокого уровня.

Нельзя сказать, какой язык самый современный. Многие из них используются очень широко и активно. Но если речь идет о перспективных языках программирования, то лучшими можно назвать такие:

- Java. Наиболее универсальный и популярный язык программирования, с помощью которого можно разрабатывать как приложения для компьютеров, так и для гаджетов, особенно под управлением Android ОС. Он имеет понятный синтаксис, учится довольно легко и быстро, первые программы на Джава можно писать уже спустя несколько недель после начала обучения. Главная характеристика, которая делает его очень перспективным – использованием для программирования под Android, который сейчас развивается очень бурно;
- C#. Отличный вариант для человека, который хочет посвятить свою деятельность написанию программ под компьютерные системы. Именно язык СИ Шарп (C#) является той основой, на которой пишется большинство программ для различных платформ и сервисов от Microsoft. С его помощью можно как разрабатывать веб-приложения с применением .NET и Azure, так и программы непосредственно для ОС Windows, различных приложений для бизнеса и многих других вещей. Чтобы разобраться в C#, придется немного попотеть, но в итоге это окупится;
- PHP. Если Ваша задача – писать качественные скрипты и интерактивные шаблоны для интернет ресурсов или быть администратором какого-либо сервера, что сейчас является очень выгодным занятием, то PHP будет лучшим способом реализовать подобное стремление. Современные языки программирования просто не могут обойтись без этого представителя. Учится он довольно таким быстро и легко – если изучать PHP тщательно и регулярно,

то уже через 2-3 года можно претендовать на очень неплохую должность и соответствующую зарплату.

Конечно, это далеко не полная характеристика языков и их классификация, но даже такого небольшого списка вполне достаточно, чтобы выбрать для себя хорошую и перспективную сферу деятельности.

Полный обзор современных языков программирования мог бы занять слишком много времени, но это не столь необходимо, так как стоит помнить главное – человек, хорошо владеющий хоть каким-либо языком, обязательно сможет найти свое место в сфере программирования. А знания одного из указанных языков обязательно будет достаточно, чтобы хорошо зарабатывать и решать интересные задачи.

2. Виды языков программирования высокого уровня

2.1 процедурно-ориентированные языки

Процедурно-ориентированные языки программирования относятся к машинно-независимым. Они являются основными языками описания алгоритмов и имеются в математическом обеспечении по существу всех современных вычислительных машин. Операционная система EG ЭВМ позволяет использовать при программировании такие языки, как Алгол, Фортран, Кобол и ПЛ / 1, относящиеся к этой группе. Будучи почти независимыми от конкретной вычислительной машины, они приближаются по синтаксису к естественным языкам. [2]

Процедурно-ориентированные языки программирования (в дальнейшем назовем их просто процедурными) могут лишь косвенно отражать содержание будущей реализации, используя специфические синтаксические конструкции для описания порядка преобразований. [3]

Процедурно-ориентированные языки, эти те которые употребляются для записи процедур или алгоритмов обработки информации на любом круге задач:

а) язык Фортран (Fortran) (от Formulae Translation — «преобразование формул»). Фортран является одним из старейших языков программи-

рования высокого уровня. Его существование и применение объясняется простотой его структуры;

б) язык Бейсик (Basic), который можно расшифровать как «Beginner’s All-purpose Symbolic Instruction Code» (BASIC) — «многоцелевой симво-

лический обучающий код для начинающих», применяется с 1964 г. как язык для обучения программированию;

в) язык СИ (C), используется с 1970-х гг. как язык системного программирования специально для написания операционной системы UNIX. В 1980-е гг на основании языка С разработали язык С++, который включает в себя язык С и дополнен средствами объектно-ориентированного программирования;

г) язык Паскаль (Pascal) получил свое название в честь французского ученого Б. Паскаля. Его начал применять с 1968—1971 гг. Н. Вирт. При создании Паскаль использовали для обучения программированию, но впоследствии он стал применяться для разработки программных средств в профессиональном программировании;

Различают такие языки процедурного программирования:

Язык **Фортран** создан в начале 50-х годов 20-го века для программирования научно-технических задач;

Кобол – создан в конце 60-х годов 20-го века для решения задач обработки больших объемов данных, хранящихся на различных носителях данных;

Алгол (1960 год) – это многоцелевой расширенный язык программирования. В нем впервые введены понятия “блочная структура программы” и “динамическое распределение памяти”;

В середине 60-х годов 20-го века был создан специализированный язык программирования для начинающих – **BASIC**. Характеризуется простотой освоения и наличием универсальных средств для решения научных, технических и экономических задач, а также задач, например, игровых.

Все перечисленные выше языки были ориентированы на различные классы задач, но они в той или иной мере были привязаны к конкретной архитектуре ЭВМ.

В 1963-1966гг был создан многоцелевой универсальный язык **PL-1**. Этот язык хорошо приспособлен для исследования и планирования вычислительных процессов, моделирования, решения логических задач, разработки систем математического обеспечения.

Язык **Паскаль** (PASCAL) (1968-1971гг)- язык процедурного программирования наиболее популярный для ПК, который и в настоящее время успешно применяется. В основу языка Pascal положен подход от общей задачи к частным (более простым и меньшим по объему). К основным принципам, которыми обладает Паскаль, можно отнести: а) Структурное программирование, которое основано на использовании подпрограмм и независимых структур данных; б) Программирование “сверху-вниз”, когда задача делится на простые, самостоятельно решаемые задачи. Затем выстраивается решение исходной задачи полностью сверху вниз.

К языкам процедурного программирования можно отнести язык **АДА** (1979 г) Язык назван в честь первой программистки Ады Лавлейс- дочери Байрона. Его отличает модульность конструкций.

Язык **СИ** (начало 70-х годов) также относится к языкам процедурного программирования. Первоначальный его вариант планировался как язык для реализации операционной системы Unix вместо языка Ассемблера. Одной из особенностей языка СИ является то, что различия между выражениями и операторами сглаживаются, что приближает его к функциональным языкам программирования.

2.3 Объектно-ориентированные языки

Объектно-ориентированные языки программирования пользуются в последнее время большой популярностью среди программистов, так как они позволяют использовать преимущества объектно-ориентированного подхода не только на этапах проектирования и конструирования программных систем, но и на этапах их реализации, тестирования и сопровождения.

Объектно-ориентированное программирование (ООП) - это парадигма (совокупность понятий и идей) программирования, в рамках которой «во главу угла» ставят понятия объектов и классов. Сейчас ООП так или иначе присутствует во всех языках, поэтому понимание его основ просто необходимо для всех, кто собирается заняться программированием.

Стоит сразу определить базовые понятия класса и объекта:

- **Класс** - это шаблон, описание ещё не созданного объекта. Класс содержит данные, которые описывают строение объекта и его возможности, методы работы с ним;
- **Объект** — экземпляр класса. То, что «рождено» по «чертежу», то есть по описанию из класса. В качестве примера объекта и класса можно привести технический чертёж для изготовления детали — это класс. Выточенная же на станке по размерам и указаниям из чертежа деталь - объект.

Появление **объектно-ориентированного программирования** стало результатом возросших требований к функционалу программ, когда описывать объект приходилось раз за разом в разных участках кода. Тогда и было введено понятие класса, параметры которого задавались единожды, а после в коде оставлялись только ссылки на класс, чтобы код самостоятельно «собрал» объект. На ООП-языке Java описание класса выглядит так:

```
// Описываем отдельный новый класс
class Circle {
// свойства класса
public double x; // абцисса центра
public double y; // ордината центра
public double y; // радиус
// методы класса
// выводит на экран параметры окружности
public void printCircle(){
System.out.println("Окружность с центром (" + x + ";" + y + ") и радиусом " + r);
}
// перемещает центр, движение окружности
public void moveCircle(double a, double b){
x = x + a;
y = y + b;
}
// масштабируем, выполняем преобразование подобия с коэффициентом k
public void zoomCircle(double k){
r = r * k;
}
}
```

Это код для окружности с заданными параметрами. Позднее при написании программы функцию вызова окружности можно будет внедрить с помощью добавления класса Circle, а не описывать его заново, что сильно экономит время, если, к примеру, мы пишем приложение для решения геометрических задач, где предусматриваются окружности разных диаметров и другие фигуры.

В качестве более живого примера **объектно-ориентированного программирования** можно привести мастерскую, где есть старший слесарь (программист) и ученики (разные участки кода). При устаревших парадигмах программирования старшему слесарю пришлось бы сначала объяснить ученику №0, как вырезать деталь (создать объект), затем ученику №1 то же самое, потом ученику №2 и так далее. ООП же даёт слесарю целую пачку бесконечных подробных чертежей (классов) деталей, которые он может раздавать ученикам вместо повторного объяснения (заведения описания объекта). Как нетрудно догадаться, это ускоряет работу и позволяет старшему слесарю уделить внимание более важным проблемам мастерской.

Объектно-ориентированное программирование выделяется не только описанной выше системой классов, вернее, её особенность не только в сохранении большого количества параметров. При упущении одного параметра, код приходилось перерывать от и до в поисках ошибки. Поэтому был разработан поведенческий аспект, что означало, что отныне классы могут не только служить вместилищем для данных, но и сами могли бы работать с ними: загружать, сохранять, изменять и выполнять другие операции.

В **объектно-ориентированном программировании** выделяют 4 основных принципа: абстракция, инкапсуляция, наследование и полиморфизм.

Понятие абстракции в объектно-ориентированном программировании

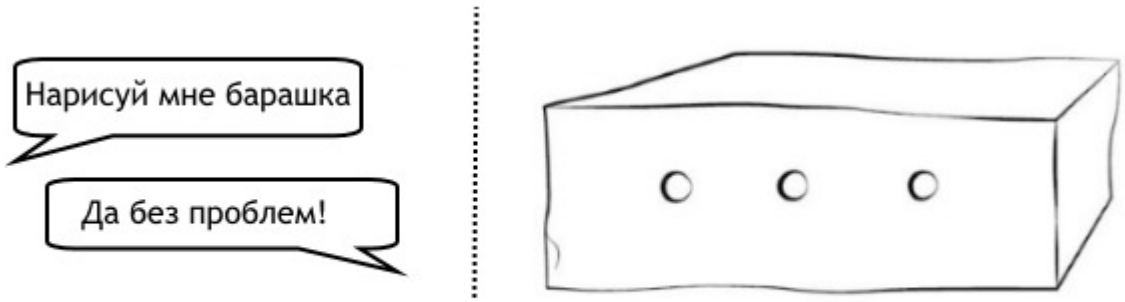
Абстракция — способ выделения самых значимых характеристик объекта, при этом менее значимые отбрасываются. В ООП абстракция - работа только со значимыми характеристиками. Суть этого принципа в том, чтобы отделить составные объекты, состоящие из «меньших» объектов, от этих самых объектов, то есть от их составляющих.

Такой подход позволяет работать непосредственно с объектом, не вдаваясь в подробности, из чего же он состоит и как работает. Возвращаясь к примеру про слесарную мастерскую, принцип абстракции заключается в том, что старший слесарь не тратит своё время и ресурсы на определение, из чего ученик сделал

деталь, а просто использует её по назначению.

Принцип инкапсуляции в ООП

Инкапсуляция - принцип **объектно-ориентированного программирования**, позволяющий собрать объект в пределах одной структуры или массива, убрав способ обработки данных и сами данные от «чужих глаз».



Инкапсуляция позволяет скрывать детали реализации

Это одновременно и облегчает конечному пользователю работу с программой, и защищает данные и само приложение от постороннего вмешательства. Пользователь может работать со всем функционалом через интерфейс, не задумываясь над тем, как программа работает. Инкапсуляцию применяют:

- когда нужно сохранить некоторый участок кода без изменений со стороны пользователя;
- когда нужно ограничить доступ к коду - в связи с уникальностью используемых техник, которые автор хочет оставить «при себе»;
- когда изменение кода повлечёт за собой неработоспособность программы или её взлом.

Наследование классов в ООП

Наследование — способность в **объектно-ориентированном программировании** построить новый класс на основе уже заданного. При этом функционал может как полностью совпадать, так и отличаться. Класс-донор называется в таком случае родительским или базовым, а его «потомок» — наследником, дочерним классом.

Существует также множественное наследование, при котором у класса-наследника может быть несколько «родителей». При этом класс наследует методы всех своих отцов и матерей, что часто приводит к ошибкам. Наследование требует определения ещё одного понятия:

- **прототип** — объект-образец, на основе которого «рождаются» другие объекты, полностью копируя его или изменяясь в процессе. При изменении в прототипе в копиях также происходят соответствующие изменения.

Принцип полиморфизма

Полиморфизм — способность объектов самим определять, какие методы они должны применить в зависимости от того, где именно в коде они находятся. То есть, объект может изменяться в зависимости от своего местоположения и действовать по-разному, что позволяет не заводить лишних структур. Иначе говоря: один интерфейс — множество решений.

Полиморфизм - это один интерфейс для множества реализаций

Полиморфизм позволяет повысить процент повторного использования кода и сократить тем самым размер программы и временные затраты на её написание.

На сегодняшний день самые популярные языки программирования - это объектно-ориентированные, к примеру, C++ и Java. Также существуют языки, которые не предполагают написания программных операторов, а программирование происходит в виде визуального проектирования с помощью интерфейса языка. Примером таких систем являются VisualBasic, Delphiи C++ Builder.

Итак, далее мы раскроем различие между процедурно ориентированные языки и объектно-ориентированные языки.

Пожалуй, каждый, кто хоть немного интересуется программированием, знает, что при разработке ПО и web-приложений используются 2 основных подхода — процедурный и объектно-ориентированный. Сразу скажем, что ни один из них не является хорошим или плохим. Это — разные способы организации кода, разные способы решения задач. Да и задачи, собственно говоря, они решают тоже разные.

Чем же отличается ПП от ООП? Какие преимущества и недостатки они имеют? И, наконец, какой из этих подходов следует изучать начинающему программисту?

Различия между процедурным и объектно-ориентированным программированием

Представь: тебе нужно написать небольшую программку, которая запрашивает у пользователя слово, считает количество символов в нем и выводит результат на экран (уже страшно, правда? ;-)).

В этой простой задаче можно выделить несколько подзадач: запросить информацию, поместить ее в переменную, посчитать количество символов с помощью специального метода и вывести результат на экран. Кроме того, следует организовать проверку корректности полученной информации и сообщить пользователю, что он ввел не текст, а, например, число или вовсе оставил форму пустой.

Все эти подзадачи называются *процедурами*. При работе в процедурном ЯП разработчик разбивает общую задачу на более мелкие и выбирает языковые конструкции для их реализации. Этими конструкциями являются ветвления, циклы, функции и другие структурные операторы.

А теперь другой пример. Допустим, ты продаешь автомобили (респект!). Каждую новую машину, предназначенную для продажи, ты вносишь в каталог через специальную форму на сайте. Форма содержит следующие поля:

- Марка авто
- Модель авто
- Мощность двигателя
- Цвет
- Год выпуска

Разумеется, если ты продаешь только легковые машины, то здесь можно применить процедурный подход — продумать алгоритм, разбить задачу на несколько шагов и написать скрипт.

Но как быть, если кроме легковых машин, в каталоге находятся, например, тракторы? Да, они, как и автомобиль, имеют марку, модель, показатель мощности, но в то же время отличаются некоторыми другими характеристиками, например, тяговым усилием. У обычного автомобиля такой пункт в техпаспорте отсутствует.

Какой подход выберет программист в этом случае? Безусловно, объектно-ориентированный. Работа сведется к следующему: создается базовый класс «Техника», в котором будут храниться характеристики, общие и для легковых авто, и для тракторов. Затем создаются два объекта — «Легковой автомобиль» и «Трактор», которые *наследуют* все характеристики из класса «Техника», а затем дополняются уникальными данными — тем самым «тяговым усилием» и пр.

Таким образом, в основе объектно-ориентированного программирования лежит понятие «объект». Иначе их называют экземпляры класса, и это вполне логично, учитывая, что они многое наследуют у класса. Кстати, наследование — это один из главных принципов ООП, наряду с полиморфизмом и инкапсуляцией. Но это уже совсем другая история.

Преимущества и недостатки процедурного программирования

Увы, все в этом мире имеет свои минусы и плюсы. Среди недостатков ПП можно назвать следующие:

- Риск возникновения множества ошибок при работе над большим проектом. Приходится писать много процедур, и это не может не сказаться на чистоте и работоспособности кода.
- Все данные процедуры доступны только внутри нее. Их нельзя вызвать из другого места программы и при необходимости придется писать аналогичный код. А это уже противоречит одному из основополагающих принципов программирования, который звучит как Don’t Repeat Yourself (Не повторяйся).
- Сложность изучения для начинающих. Этот недостаток может кому-то показаться притянутым за уши, но простая статистика свидетельствует, что процедурное программирование для *большинства* новичков дается сложнее, чем объектно-ориентированное.

Впрочем, у ПП есть и свои преимущества. Среди них отметим:

- Любая процедура (функция) может быть вызвана неограниченное количество раз. Все как в жизни — ты один раз «написал» в голове маршрут к любимой пиццерии, а затем просто вызываешь эту «программу» из памяти.
- Возможность оперативно решить задачу, в которой отсутствует сложная иерархия. Можно пойти дальше и сказать: если проект не подразумевает создания большого количества классов и объектов, то в ПП совсем нет минусов.

Преимущества и недостатки объектно-ориентированного программирования

Главным минусом использования ООП можно назвать громоздкость при решении простых задач. Сравни, например, два участка кода, написанного на PHP. Первый пример — процедурный код, второй — объектно-ориентированный. И тот, и другой скрипт ведут к одному результату: просто выводят на экран фразу «Hello, world»:

Скрипт №1

```
<?php

print «Hello, world.»;

?>
```

Скрипт №2

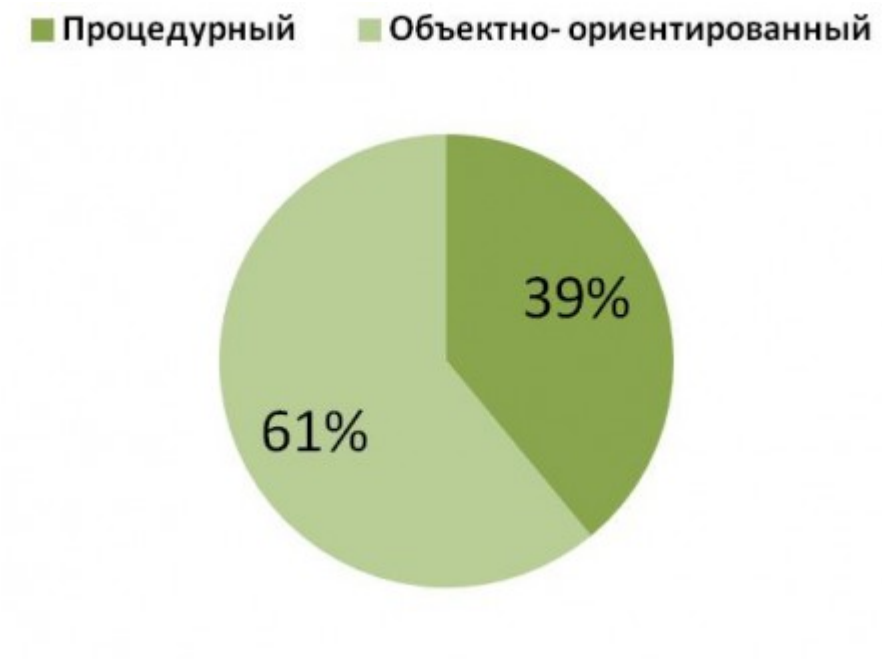
```
<?php

class helloWorld {
function myPrint() {
print «Hello, world.»;
}
}
$myHelloWorld = new helloWorld();
$myHelloWorld->myPrint();

?>
```

Показательно, не правда ли?

С другой стороны, у ООП есть очень большой плюс: такой код удобнее поддерживать, изменять и обслуживать, так как он разбит на модули, которые проще воспринимаются визуально. Да, и ошибок меньше.



Кроме того, объектно-ориентированные языки программирования легче изучаются новичками. Взгляни на диаграмму — она отражает результаты анкетирования студентов, которые познакомились с обоими подходами к разработке. Как видно, большинство из них выбрало именно ООП для более детального изучения.

Подводим итоги. Не существует плохого или хорошего ЯП. А вот плохие программисты встречаются, и даже очень часто. Хороший разработчик умеет «помирить» два подхода к программированию в своем сознании и использует оба.

Помни: язык — это всего лишь инструмент. Он не должен управлять программистом так же, как хвост не может вилять котом. Если же это происходит, то виноват программист, но не хвост, ведь так?

При реализации того или иного проекта разработчик сам решает, как он будет реализован. Если перед тобой стоит серьезная задача, и ты понимаешь, что без классов и объектов здесь не обойтись — выбирай ОО-язык. К ним относятся Delphi, Java, C#, JavaScript.

Если же задача довольно проста и ты четко видишь *пошаговый* алгоритм ее решения, то твоя стихия — это процедурное программирование. К процедурным языкам относятся Basic, Pascal, C.

2.3проблемно-ориентированные языки

В настоящее время вслед за теоретиками в области информационных технологий (ИТ) многие разработчики программного обеспечения сходятся во мнении, что механизм абстракции является одним из ключевых факторов, определяющих качество и эффективность реализации сложного программного продукта. Своего рода «венцом» целого ряда исследований по тематике *абстрактных типов данных* в 70-х годах прошлого века явился язык *CLU,* предложенный Барбарой Лисков [1]. Несомненна концептуальная значимость языка *CLU* - он определил одну из основ *объектно-ориентированного подхода.* В то же время продукт Б. Лисков не получил сколько-нибудь серьезного распространения в среде профессиональных разработчиков, прежде всего, вследствие излишней академичности и перегруженности специфическими конструкциями.

Достаточно очевидным является то, что разумно примененный механизм абстракции позволяет весьма эффективно осуществлять командную разработку, поддерживая при этом высокое качество программного продукта и эффективное управление процессом. Отложенные вычисления [2-6], модули [2-6], объекты [6, 7], манданты [5, 8] - все это лишь некоторые из современных инструментов введения и поддержки абстракции. Разные языковые среды поддерживают эти механизмы в той или иной степени, одни хорошо, другие не очень. Общеизвестно, что любые алгоритмические решения могут быть реализованы в рамках различных существующих парадигм программирования (выбор, в значительной степени, определяется личными предпочтениями). Закономерно возникает вопрос, что же может претендовать на роль «идеального» механизма абстракции при написании приложений, лежит ли он в плоскости технических средств (подобно упомянутому выше *CLU)* или в большей степени определяется соответствующей методологией, используемой в процессе работы с традиционным инструментарием.

Следуя сложившейся традиции [9-11], определим ПОЯ *(DSL — Domain Specific Language)* следующим образом: язык программирования, который обладает ограниченной степенью выразительно- сти/силы, сфокусированный на определенной предметной области и содержащий конструкты, отражающие абстракции именно этой предметной области.

Проблемно-ориентированные языки принято разделять на три группы [9]:

1. *Внешние (external DSL).*
2. *Внутренние (internal DSL).*
3. Инструментальные средства языка *(language workbench).*

public void insert(User aUser) throws Exception { aUser.setGeneration(O);

getTemplate.update("insert into user_table (id, created, properties, generation, status, backend, email, pswd) values (?,?,?,?,?,?,?,?)"), aUser.getID(), aUser.getCreated(),

Bytes.toByteArray(aUser.getProperties()),

aUser.getGeneration(),

aUser.getStatus().toString(),

aUser.getBackendID(),

aUser.getEmail(),

aUser.getPassword());

}

Рис. 1. Использование внешнего ПОЯ (SQL) из языка Java

Рассмотрим эти группы подробнее.

1. *Внешние ПОЯ* используют синтаксис, полностью отличный от синтаксиса языка основного приложения. Единственным исключением, пожалуй, является ситуация, когда в качестве основного языка берется *XML*. Внешние ПОЯ - это наиболее яркие представители современных инструментов введения абстракции: *little languages* в *Unix,* регулярные выражения, *SQL, PostScript, awk, HTML* и т. д.

Самый большой плюс внешних ПОЯ состоит в том, что их можно писать, не оглядываясь на стандарты и спецификации. Иначе говоря, разработчик может выразить предметную область в самой лаконичной и пригодной для чтения и редактирования форме. Качество такого ПОЯ определяется лишь умением разработчика создать транслятор, который сможет откомпилировать входной файл и выдать исполняемый код - как правило, уже на основном языке приложения. Отсюда же следует и очевидный недостаток внешних ПОЯ - необходимость создания этого самого транслятора.

Еще один существенный недостаток внешних ПОЯ определяется отсутствием у них того, что принято называть «символической интеграцией» *(symbolic integration)* [12]. Внешний ПОЯ, на самом деле, никак не связан с основным языком прило-

жения. Программная среда разработки для базового языка, на котором пишется это приложение, не содержит информации о новом ПОЯ, и следовательно, проблема редактирования связей исполняемого кода стоит довольно остро.

Обратимся к примеру, приведенному на рис. 1. Предположим, разработчик решил изменить свойства целевого класса *User*. Во всех современных средах разработки автоматическим переименова- нием/рефакторингом уже никого не удивишь. Однако это переименование не будет работать в *SQL* коде. Это и есть тот самый «символический барьер» между *Java* и *SQL* - он не позволяет манипулировать собранной программой как единым целым.

1. *Внутренние ПОЯ* используют существующий (обычно - универсальный) язык как свою основу (последний часто называют языком-носителем - *host language*). При этом создаваемый ПОЯ, как правило, получается синтаксически совместимым с языком-носителем и может быть обработан, ис- пользуя его инфраструктуру (скомпилирован штатным компилятором, интерпретирован, отлажен и т. д.). Полученный таким образом ПОЯ является одновременно как *расширением* языка-носителя, так и его *ограничителем.*
2. При построении внутреннего ПОЯ как *расширения* (рис. 2, а), нововведенные концепции становятся доступными для языка-носителя, и окончательным результатом является новый язык, который обладает полным функционалом исходного языка и добавляет специфичные расширения.*ие внутреннего ПОЯ: а) расширение, б) сужение*

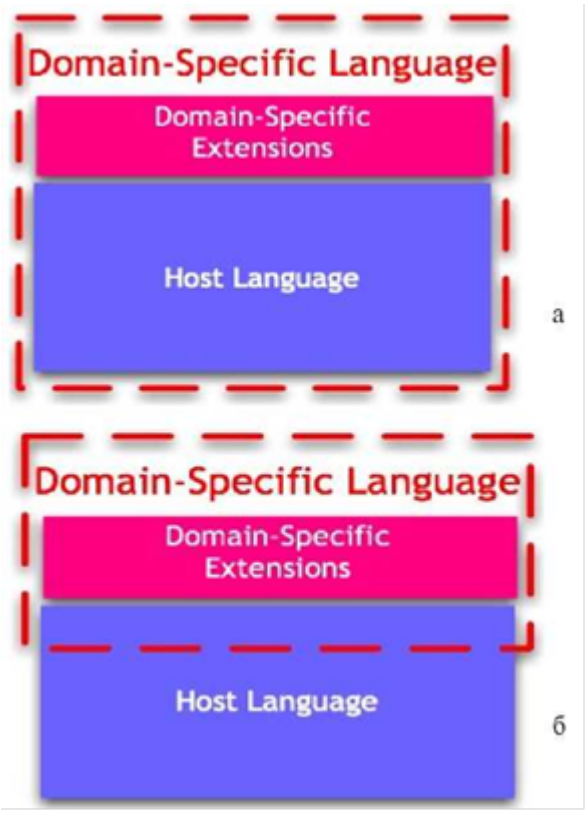


Рис. 2. Построен

1. В случае реализации внутреннего ПОЯ как *ограничителя* языка-носителя (рис. 2, б), новый язык отражает специфику предметной области, при этом скрывая (а зачастую, запрещая) большинство конструкций языка-носителя, которые не имеют отношения к задачам данной предметной области. Окончательный результат в таком случае - это также новый язык, только, в некотором смысле, суженный.
2. К классическим примерам внутренних ПОЯ можно отнести: *LISP* (опытные LTSP-программисты говорят о процессе программирования на *LISP,* как о постоянном создании и использовании новых ПОЯ), *Ruby* и *JRuby* (большинство библиотек *Ruby* выполнены в стиле ПОЯ, как например, *Rails)* и ряд других языков. Пример использования внутреннего ПОЯ при работе с *Ruby* приведен на рис. 3.

```
class XmlBookDetailModel

#implementing IBookDetailModel.java interface include IBookDetailModel

@doc

def initialize {...}
```

Methods implementing IBookDetailModel interface ### def getAllBooks {...}

def loadBookDetail(book)

isbn = book.getIsbn

#find matching book by isbn @doc.elements.each("books/book") { |elem| if (elem.attributes['isbn'] == isbn)

book.setTitle(elem.attributes['title']) book.setAuthor(elem.elements['author'].text) book.setPublisher(elem.elements['publisher'].text) book.setDatePublished(elem.elements['datePublished'].text) book.setDescription(elem.elements['description'] .text) return book end

}

return book end end

Рис. 3. Пример использования внутреннего ПОЯ - Ruby

Плюсы и минусы внутренних ПОЯ являются, практически, зеркальным отражением внешних. Здесь уже нет символьного барьера. Разработчик может в полную силу пользоваться возможностями языка-носителя и задействовать все инструменты, которые есть для этого языка.

Классический пример иллюстрируется рис. 4. Использование конфигурационного файла для определения параметров работы системы может быть упрощено, если в качестве формата конфигурационного файла выступает программа, выполняемая системой, например, при загрузке. Пример демонстрирует использование языка Ruby, как основы ПОЯ, при этом собственно Ruby-код в примере не содержится. Язык-носитель ограничен до минимальных требований задачи, включая изменение парадигмы императивного программирования на декларативную.

describe node{

ip_[200,122,122,122] hostname ['admin-node'] system port 3000 alternative port 3001 security method :SSL

}

describe node{

ip_[200,122,122,102] hostname ['indexer-node'] system port 3000 alternative port 3001 security method :none java version JDK 1 4

}

Рис. 4. Пример использования ПОЯ для конфигурирования системы

Отметим, что возможность использования для ПОЯ всей мощи языка-носителя обладает как достоинствами, так и недостатками. Характерной особенностью ПОЯ является возможность работы без изучения всех нюансов носителя. Это призвано обеспечить возможность экспертам предметной области (а не профессиональным программистам) вносить алгоритмику предметной области непосредственно в программную систему. Внутренний ПОЯ может усложнить этот процесс, поскольку существует множество мест, где пользователь может оказаться в тупике из-за недостаточной осведомленности о возможностях базового языка.

1. Инструментальные средства языка (language workbenches) - это среды разработки (IDE - Integrated Development Environment), предназначенные для создания новых ПОЯ. Среда позволяет определять абстрактный синтаксис языка совместно с редакторами и генераторами языка (см. рис. 5). Редактор позволяют получить продвинутое и удобное окружение, которое учитывает специфику создаваемого ПОЯ по аналогии с IDE для языков общего назначения (Eclipse, MS Visual Studio, IDEA, и т. д.).

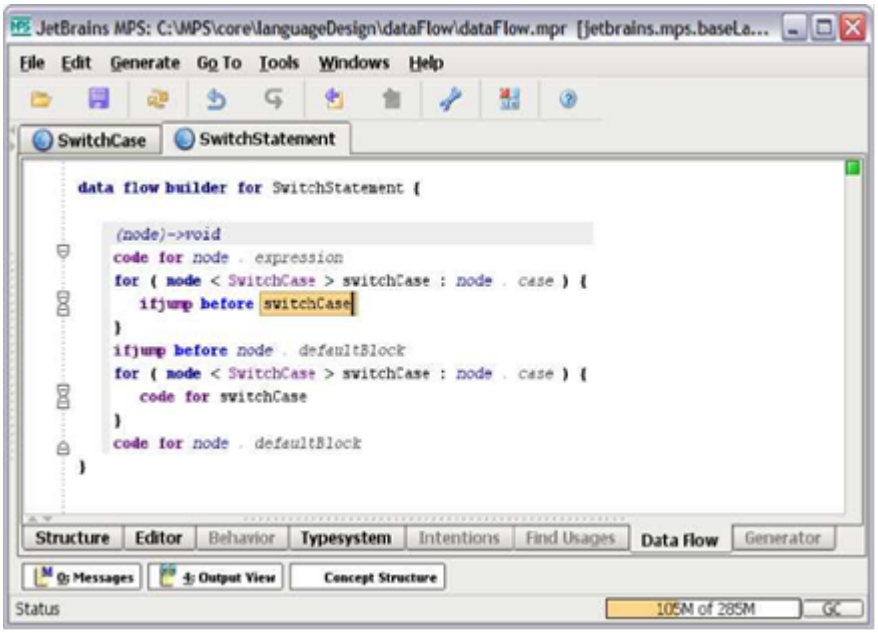


Рис. 5. JetBrains MPS - пример оболочки для разработки ПОЯ

При использовании подобного инструментария разработчик, практически, не пишет код, а лишь манипулирует абстрактным представлением программы. Разумеется, IDE отображает эти изменения в тексте программы, но сути это не меняет - разработчик модифицирует не код, а абстрактное представление. Нечто подобное наблюдается и во время процесса рефакторинга.

О целесообразности разработки ПОЯ

Проблемно-ориентированный язык остается довольно специфичным инструментом - он не вполне вписывается в объектно-ориентированную методологию разработки или в другие модели проектирования (такие как, например, Agile Processes), которые представляют собой существенный сдвиг в понимании процесса разработки. Парадигма ПОЯ предоставляет разработчику возможность в одном и том же проекте использовать 5-10 разных языков, в зависимости от задач конкретного моду- ля/подсистемы без изменения общего подхода к реализации.

Можно выделить следующие достоинства и недостатки ПОЯ, как опорные точки при принятии решения об использовании в ИТ-проекте обсуждаемой парадигмы.

Достоинства:

• Использование труда экспертов — это возможность исправить то состояние дел, которое наблюдается сейчас в области взаимодействия программистов и экспертов в предметных областях. Последнее является главным камнем преткновения на пути успешной работы над проектами.

• Смена контекста выполнения - даёт разработчику широкие возможности по оптимизации продукта.

• Использование альтернативных моделей вычислений в рамках одного проекта - позволяет гибко комбинировать парадигмы программирования (например, объектно-ориентированные и функциональные).

• Резкое снижение уровня «семантического шума» - позволяет разработчику сконцентрироваться на решении проблемы предметной области.

Недостатки:

• Высокая стоимость разработки, миграции и поддержки - ПОЯ, как и любой продукт, отходящий от господствующих тенденций, отличается от традиционных конкурентов и более высокой ценой.

• Высокие требования к архитектуре ПОЯ- сильно ограничивают спектр предложений на рынке труда.

• Языковое смешение в рамках одного проекта/продукта - значительно усложняет процесс детального понимания принципов работы системы.

• Постепенное перерождение ПОЯ в язык общего пользования - достаточно трудно сохранять баланс между расширениями ПОЯ и конструкциями языка общего пользования (примером трансформации языка для обработки текстовых данных в язык общего пользования может служить AWK).

Заключение

Подводя итоги по курсовой работе, отметим, что язык программирования высокого уровня Паскаль обладает большими возможностями для решения достаточно широкого круга задач.

Вместе с тем, интуитивно понятный синтаксис, четкая структуризация, доброжелательный интерфейс среды разработки делает его одним из наиболее популярных языков программирования.

В первой главе курсовой работы нами была произведена классификация языков программирования, рассмотрены сферы их применения, вкратце изложены их основные особенности, достоинства и недостатки.

Одним из важнейших факторов при выполнении поставленной задачи является верный выбор языка программирования. Для реализации поставленной задачи можно использовать различные языки высокого уровня, наиболее распространенными из которых являются C++ и Паскаль.

Язык Паскаль является статически типизированным, компилируемым, поддерживает низкоуровневую работу с памятью. Именно поэтому его можно рекомендовать в качестве языка программирования для изучения работы подпрограмм, процедур и функций. Простота реализации графического интерфейса позволяет наглядно продемонстрировать процесс создания пользовательского меню.

Проанализировав все выше изложенное, нами был сделан аргументированный выбор языка программирования высокого уровня Паскаль для дальнейшего изучения предмета исследования.

Во второй главе курсовой работы нами были рассмотрены основные виды языков программирования высокого уровня - процедур и функций в алгоритмическом языке Паскаль. Также затронута смежная тема использование модулей для лучшей организации блочного использования процедур и функций. Приведены примеры программной реализации описанных структур.

Рассмотренные нами теоретические сведения были применены для реализации практической части курсовой работы.

Результатом выполнения практической части курсовой работы стала разработка полнофункциональной программы, на примере создания которой были наглядно продемонстрированы преимущества использования пользовательских меню при разработке интерфейса пользователя, а также возможности использования подпрограмм в языке высокого уровня Паскаль.

Таким образом, в ходе написания третьей главы курсовой работы были на практике исследованы теоретические аспекты, рассмотренные во второй главе. Был описан интерфейс пользователя разработанной программы, а также ее программная реализация. Особое внимание было уделено процессу реализации пользовательского меню.

Список использованных источников

1. Абрамов В.Г., Трифонов Н.П., Трифонова Г.Н. Введение в язык Паскаль. - М.: Наука, 2012
2. Абрамов С.А., Гнездилова Г.Г., Капустина Е.Н., Селюн М.И. Задачи по программированию. М.: “Наука”, 2013
3. Вирт Н. Алгоритмы и структуры данных. - М.: Мир, 2014
4. Вайсфельд, М. Объектно-ориентированное мышление / М. Вайсфельд. - М.: Питер, 2014. - **338** с.
5. Васильев, А. Н. Java. Объектно-ориентированное программирование / А.Н. Васильев. - М.: Питер, 2012. - 398 с.
6. Васильев, А. Н. Java. Объектно-ориентированное программирование / А.Н. Васильев. - М.: Питер, 2013. - 400 с.

7. Васильев, Алексей C#. Объектно-ориентированное программирование / Алексей Васильев. - М.: Питер, 2012. - 320 с.
8. Дагене В.А., Григас Г. К., Аугутис К.Ф. 100 задач по программированию. - М.: Просвещение, 2013
9. Дунаев В. В. HTML, скрипты и стили. Спб.: БХВ – Петербург, 2011 – 816 с.
10. Епашников А.М., Епашников В.А. Программирование в среде Турбо Паскаль 7.0. - М.: МИФИ, 2014
11. Комлев, Николай Юрьевич Объектно Ориентированное Программирование. Хорошая книга для Хороших Людей / Комлев Николай Юрьевич. - М.: Солон-Пресс, 2014. - **770** с.
12. Лафоре, Р. Объектно-ориентированное программирование в C++ / Р. Лафоре. - М.: Питер, 2015. - 928 с.
13. Лесневский, А. С. Объектно-ориентированное программирование для начинающих (+ CD-ROM) / А.С. Лесневский. - М.: Бином. Лаборатория знаний, 2010. - 232 с.
14. Мэтью Д. HTML5. Разработка веб-приложений. М.: Рид Групп, 2012 – 320 с.
15. Никсон Р. Создаем динамические веб-сайты с помощью PHP, MySQL и JavaScript. Спб.: Питер, 2011 – 496 с.
16. Павловская, Татьяна C/C++. Процедурное и объектно-ориентированное программирование. Учебник / Татьяна Павловская. - М.: Питер, 2015. - 496 с.
17. Пьюривал С. Основы разработки веб-приложений. СПб: Питер, 2015 – 272 с.
18. Рассел, Джесси Аспектно-ориентированное программирование / Джесси Рассел. - М.: VSD, 2013. - **942** с.
19. Санников, Е. В. Курс практического программирования в Delphi. Объектно-ориентированное программирование / Е.В. Санников. - Москва: **Наука**, 2013. - 188 с.
20. Шакин, В.Н. Объектно-ориентированное программирование на Visual Basic в среде Visual Studio .Net / В.Н. Шакин, А / В.Н. Шакин, Г.К. Сосновиков, З. - Москва: **РГГУ**, 2015. - **118** с.